An approach to minimize useless checkpoints on distributed optimistic simulations

Ricardo Parizotto[1], Braulio Adriano de Mello[2]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

[2]Ciência da Computação – Universidade Federal da Fronteira Sul (UFFS)
Chapecó - SC - Brasil

rparizotto@inf.ufrgs.br, braulio@uffs.edu.br

Abstract. *Distributed architectures for modeling and simulation can scale the execution of large and complex models. These architectures frequently utilize checkpoint strategies to guarantee the execution of synchronous and asynchronous components. However, the complete avoidance of useless checkpoints is impractical, and it can severely decrease the simulation performance. In this paper, we present a set of metrics to identify useless checkpoints at run-time. Additionally, we extended a probabilistic decision that employs our proposed metrics to create only checkpoints with high probability to be loaded by rollback operations. The method identifies inconsistent checkpoints based on the communication patterns and granularity of the events since the last rollback. The results showed that the proposed metrics allow reducing the number of useless checkpoints without negative impacts on simulation performance and outperforms traditional probabilistic strategies in terms of rollback time.*

1. Introduction

Modeling and simulation techniques have been applied as a tool for analyzing a broad range of complex systems. While systems and its models grow, heterogeneous and distributed simulation systems have been proposed to support more complex and larger models [Reynolds 1988]. Their execution on distributed environments demands synchronization primitives to ensure the consistency of the simulation results. However, synchronizing [Jefferson 1985] components of distributed simulation systems can rise high overhead, concerning processing and message exchanges. Rollback based mechanisms are a well know strategy to repair the state of asynchronous systems after failures. This can take on many forms, for instance: (1) Coordinating the placement of checkpoint promotes saving only secure states but relies on the need of observing states of other processes to coordinate correctly [Sato et al. 2012]; (2) non-coordinated or semi-coordinate strategies have less or any interaction with another component state but are more susceptible to problems such as domino effect.

Large and complex simulations are prone to create a large number of useless checkpoints which can severely impact the processing and memory. This scenario has been motivated the proposing of probabilistic and heuristic schemes to derive dependencies of knowledge between event interactions and guide the placement of checkpoints at runtime [Kunz et al. 2012] [Fu et al. 2013]. The heuristics for checkpoints try to identify rollbacks preemptively to determine the intervals of checkpoints dynamically [Wang et al. 2009], both to minimize the number of created checkpoints and the time spent on rollbacks. However, those strategies ignore essential factors (e.g., the checkpoint category) and fail to store important checkpoints. In addition, they frequently save states that are not used during system execution. There are open challenges for proposing approaches that avoid useless checkpoints without increasing the total time of rollback operations.

To qualify these questions, we revisited the checkpoint placement problem and presented a set of metrics to measure the utility of a checkpoint at run-time. The method is based on communication patterns between dependent process composed with the granularity of events since the last rollback. We employed these measurements in a probabilistic decision that utilizes the measurements to estimate the usability of a checkpoint without the needing to observe the state of any other process. Consequently, the approach does not require additional management messages. Otherwise, it works based on information extracted from the simulation behavior. We assume that components do not share a global time and rely on the observation: if a rollback occurs to a time inferior to a checkpoint, it will not be useful to restore to a consistent state. We integrated the solution into the DCB (Distributed Simulation Backbone) [Carvalho 2015] and performed extensive simulations in a synthetic scenario. The results of experimental simulations show that the method decreased the number of useless checkpoints without increasing the time spent on rollback operations.

In the remainder of this paper, the Background section presents the central issues of distributed simulations and useless checkpoints. Then we present the specification of metrics and the probabilistic checkpoint method. Next, we give details about the implementation of our method into DCB and our case study, followed by the main related works and the concluding remarks.

2. Background and Motivation

In this section, we review the aspects of synchronization in distributed simulation architectures. We also review the classification of useless checkpoints according to the rollback scenario.

Distributed Simulation The system assumes that a distributed architecture connects independent and heterogeneous components for supporting the cooperation among them. The inner behavior of each component is seen as a black box and is connected to the core of the simulator by an interface. The interface stores a log of messages which are ordered by their timestamp (i.e., time to process the message), and associated with its respective source by a dependency vector [Johnson 1990]. The dependency vector defines the dependency among the components according to their configuration of the input and output ports. Simulation components change their states by executing events. When a process A executes an event e which will make changes on the state of the process B, a simulation message is sent from A to B. When the destination process receives the message, it executes the event e' and e precedes e' (or $e < e'$). It defines the causality effect of e over e' [Lamport 1978]. Therefore, a process P_k depends on the process P_j (denoted by $P_k \Rightarrow P_j$) if P_j is configured to send a message requesting the execution of an event e by P_k. And, if P_j depends on the P_w (denoted by $P_j \Rightarrow P_w$), by transitivity, P_k also depends on the P_w (i.e., $P_k \Rightarrow P_w$) [Netzer and Xu 1995].

Useless checkpoints Components of distributed simulations manage their own local virtual time (LVT) and must execute according to its value [Mattern et al. 1989]. A time violation (or LCC-Local Causality Constraint) happens when a process receives a message with the *timestamp* less than its LVT. When a component suffers an LCC violation, it executes rollback operations to restart in a prior consistent state previously saved in a checkpoint and then run the message in the correct order. Checkpoints are categorized into two different categories: inconsistent and unreachable. A checkpoint becomes inconsistent if in some instant a rollback is performed to a timestamp before the checkpoint. Inconsistent checkpoints turn the system susceptible to domino-effects (i.e., requires additional rollbacks to restore the system to a consistent state). Any rollback operation never restores unreachable checkpoints. This kind of checkpoint can be created by coordinate or non-coordinated strategies and depend on garbage collection techniques to be eliminated from memory [Elnozahy et al. 2002].

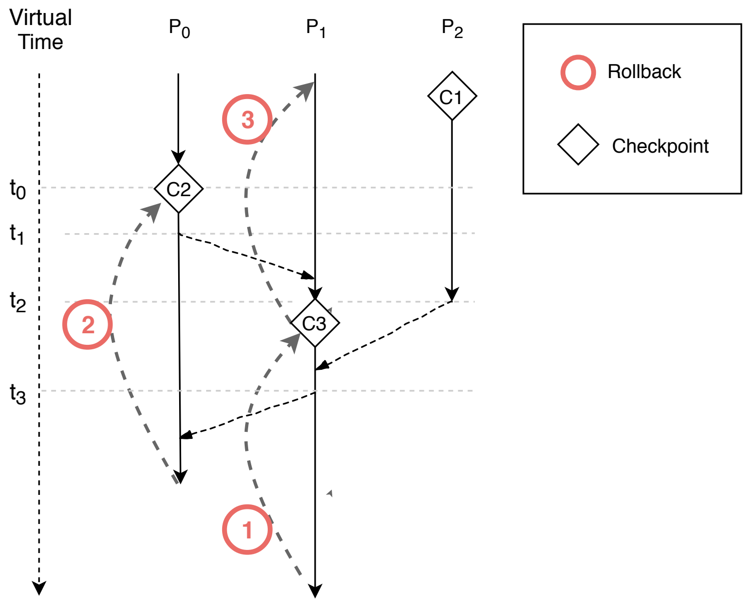

Figura 1. System state with inconsistent checkpoints

Figure 1 illustrates an LCC violation. In the exemple, process P_0 send a message to P_1 triggering a rollback to its last checkpoint (Figure 1, rollback 1). The rollback leaves a orphan message in P_0 and requires a rollback on P_0 (Figure 1, rollback 2). The rollback operation executed by P_0 generates another orphan message and it makes the global state inconsistent. Another rollback of the process P_1 eliminate the orphan messages and turn the global state consistent again (Figure 1, rollback 3), but leave C_3 as a inconsistent checkpoint.

In this work, we extend a probabilistic checkpoint approach to reduce the number of inconsistent checkpoints in distributed simulations. We assume that components do not share a global time and rely on the observation: if a rollback occurs to a time inferior to a checkpoint, it will not be useful to restore to a consistent state.

3. Probabilistic Checkpointing Method

This section presents a set of metrics to measure the utility of a checkpoint. The metrics operate by identifying communication patterns between dependent processes of the simulation. Next, we correlate the communication patterns with the rollback probability to extend a checkpoint method proposed by Quaglia [Quaglia 1999]. Finally, the method uses the correlation to estimate the usability of a checkpoint at runtime.

3.1. Metrics Formulation

Table 1 presents a description and notation of the proposed metrics. The LVT and timestamp are already available by the target architecture. The N_{events} monitors the number of events processed by the process. The $N_{rollbacks}$ monitors the number of rollbacks that the process made. The E_r is simply the number of events that a process made since the occurrence of its last rollback. These metrics works basically as event counters and have their values updated according to the event execution of each component. Further metrics are updated only on the checkpoints generation.

Tabela 1. Description and Notation of the proposed metrics

LVT_i	Local virtual time of the sender process P_i
N_{events}	Number of events executed
$N_{rollbacks}$	Number of rollbacks executed
E_r	Number of events since the last rollback
$f(P_k)$	Average time between messages received from P_k
$P_{rollback} = N_{events}/N_{rollbacks}$	Average events between rollbacks
$d(P_i, P_k) = \|LVT_i - LVT_k\|$	Temporal distance between P_i and P_k
$t(P_i, P_k) = d(P_i, P_k)/f(P_k)$	Average interval between messages

The metrics which monitor the temporal distance between two components are defined as the distance between the LVT of components. Aiming to avoid the overload of the coordination protocols, we estimate the distance between two components using the timestamp of the received messages. For instance, when calculating the distance between P_i and P_k, we assume the timestamp of the last received message from P_k as the estimated LVT of P_k. In a scenario where the LVT of a process P_i is higher than the LVT of the process P_k, and it is true that $P_i \Rightarrow P_k$, the 'distance' represents a time interval in which received messages from P_k will be LCC violation prone.

Rollback Probability We proposed the utilization of the rollback probability as a metric to create checkpoints for avoiding an excessive number of rollbacks, which can decrease the system performance. The method is applied whenever the simulation tries to generate a new checkpoint at a given simulation time. The method uses the known past events from all dependent components to predict if they can send new messages that create time violations in the components which are going to generate checkpoints.

We denote the set of allowed events on the time interval $(LVT_i - d(P_i, P_k), LVT_i)$ as Γ. Computing Γ is impractical if the distance is large. Our method approximates Γ efficiently based on a subset of received messages. It allows approximating the number of messages that would be scheduled by P_k and received by P_i with $timestamp \in \Gamma$. Applying it to each dependent process of P_i, according to equation 1, we have the number of arriving messages which may generate LCC violation on this interval.

$$\Delta P_i = \sum_{k \in DP_i} \begin{cases} \frac{d(P_i,P_k)}{f(P_u k)} & \text{if } LVT_i > LVT_k \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The measurement of $\Delta P_i \geq 1$, may suggest that there is at least one estimated message whose *timestamp* is lesser than LVT_i. In an ideal scenario, $\Delta P_i + E_r$ is the correct number of events in Γ. We then compose ΔP_i with the granularity of events since the last rollback to check if the measurement reaches the idealistic scenario. The reasoning behind that begins by assuming that a rollback is going to occur: if $\Delta P_i + E_r \geq P_{rollback}$ is true, then it indicates that more events would be scheduled in Γ than the average number of events between rollbacks, therefore the system is rollback prone; Otherwise, if there are few events since the last rollback and few messages arriving in a way that $\Delta P_i + E_r \geq P_{rollback}$ is not satisfied, by contradiction, it is probably not a rollback prone and a checkpoint must not be created.

3.2. Consolidanting simulation checkpoints

In this section, we discuss how we integrated the measurement of ΔP_i into the probabilistic checkpoint method. Aiming to avoid the overload of coordination protocols, we do not use any coordination with other components to obtain the data. The procedure uses timestamps extracted from simulation messages and utilized by the checkpointing strategy only on the checkpoints generation.

```
if  $rand() > (1 - P_{rollback})$ then
   take a checkpoint
end if
```

Quaglia first presented the probabilistic decision depicted above. [Quaglia 1999]. Whenever the system tries to create a checkpoint, the probabilistic choice is applied to determine if the current state represents a useless checkpoint. We extended the conditional clause λ with our method to measures the usability of a checkpoint, requiring that both the expressions are true to create a checkpoint [Fagin et al. 1994]. If the proposed metrics indicate that the state can be useful for restoring the system from an LCC violation, then the processes create a checkpoint. Next, we present the composition of our metrics into the probabilistic decision:

$\lambda := (rand() > (1 - P_{rollback}))...$

$\lambda := \lambda \wedge ((\Delta(P_i) + E_r) \geq P_{rollback})$

As the probabilistic decision requires the rollback probability calculation whenever it is executed, then the cost of the checkpoint creation increases proportionally to the complexity of ΔP_i, $\mathcal{O}(n)$. It happens because the dependency vector must be entirely considered by ΔP_i. The worst case is seen when the size of the vector is equal to the number n of the processes in the simulation [Bouguerra et al. 2013].

4. Implementation and Evaluation

We implemented the probabilistic method into the DCB (Distributed Co-simulation Backbone) [Carvalho 2015]. DCB is a distributed and heterogeneous simulator. Our approach was integrated into the receiver layer, that works as an interface between the core of DCB and each component. This layer keeps the messages log and the dependency vector, which are fundamental structures to calculate the rollback probability every time the system tries to create a checkpoint. In this section, we present the scenario in which we evaluated our method and the results we obtained in this scenario.

4.1. Case Study

We created a model with five asynchronous components that have the same internal behavior, and communicate with each other through simple messages. Each component generates one new simulation message each 100ms according to the configuration presented in Figure 2.

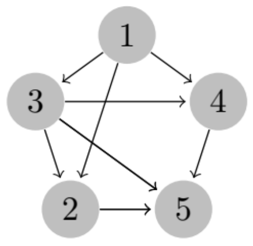

Figura 2. Message passing configuration graph

The configuration was used to execute the case study for three different checkpoint methods: Periodic, where the system generates checkpoints according to deterministic and periodic intervals; the technique proposed by Quaglia [Quaglia 1999]; and the probabilistic extension proposed in this work. We executed the experiments with these methods to compare with our probabilistic approach. We performed ten simulations of 1.000.000 units of virtual time for each one of the three methods. The experiments were performed on an Intel Core i5-3470 CPU @ 3.20GHz x 4 with 7,7 GiB of memory and running Antergos Linux 64-bit.

Evaluation It is important to measure both the number of each checkpoint category and the time spent on rollbacks, that says about the delay aggregated by the respective checkpoint strategy, in order to evaluate the checkpointing strategy. Figure 3 presents the average of checkpoints created by all the components. The experiments with the Quaglia algorithm was the one that created the least number of checkpoints. It created approximately 9% fewer checkpoints than the Periodic algorithm and 2% fewer checkpoints than the Probabilistic algorithm.

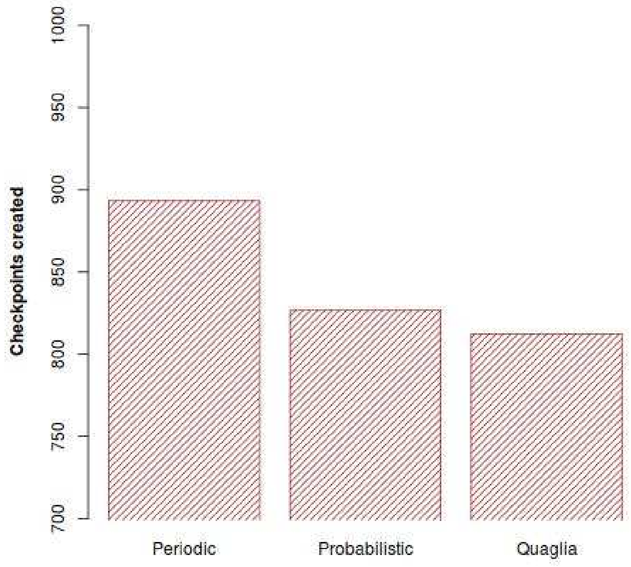

Figura 3. Average number of checkpoints created

Figure 4 presents the average of simulation time spent rollbacking. The time spent by one rollback is given by the simulation time, when happens the time violation, except the timestamp of the checkpoint restored. The Quaglia algorithm performed higher measurements for time on rollbacks. The time spent by rollbacking of the Probabilistic algorithm was 17.6% higher than the Periodic algorithm, and the Quaglia algorithm was 95.5% higher than the Periodic algorithm. It is explained the fact that the Quaglia algorithm did not create some checkpoints that would have been useful for reducing the time spent by rollbacking.

	Rollback	Inconsistent	Unreachable
Periodic	11.72	6.4	875.06
Quaglia	7.64	3.8	800.96
Probabilistic	6.48	2.92	817.32

Tabela 2. Statistics for each checkpoint method

Table 1 presents the average of rollbacks performed and the number of inconsistent and unreachable checkpoints for each checkpoint method. As a general result, the probabilistic method created less inconsistent checkpoints and consequently performed fewer rollbacks than both other strategies. The execution with the Probabilistic algorithm generated 45% fewer rollbacks than the Periodic algorithm and 15% than the Quaglia algorithm. The percents are similar to the inconsistent checkpoints. However, our algorithm generated 2% more unreachable ones than the Quaglia algorithm. Nevertheless, it spent around 35% less time executing rollback operations than the Quaglia algorithm, which we argue is an acceptable tradeoff for a system like ours.

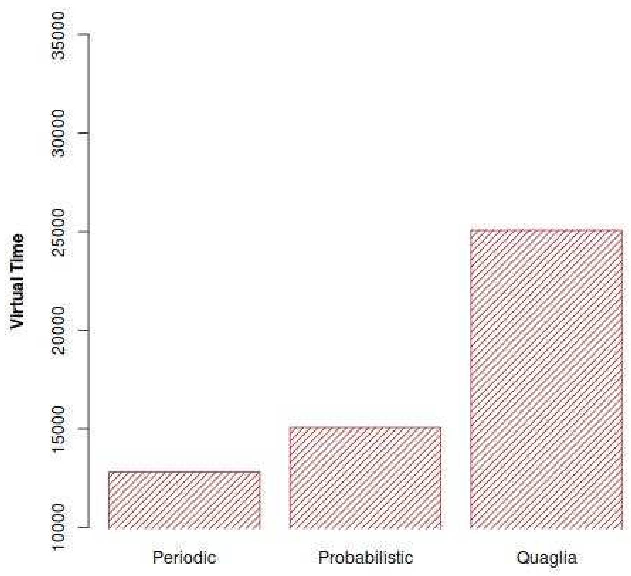

Figura 4. Average of virtual time rollbacked

5. Related Work

Several studies addressed strategies to reduce the rollback overhead while managing checkpoints efficiently. These efforts are classified as works that propose new architectures that use checkpoints and the works that propose optimization for traditional checkpoint/rollback algorithms. In this section, we review some of these strategies and highlight similarities with our proposal.

Quaglia [Quaglia 1999] proposed a checkpointing algorithm for optimistic simulations that combine periodic and probabilistic approaches. This algorithm uses the probability of the rollback occurrences in a given time interval to decide if a new checkpoint should be created at the current time. However, useless checkpoints are not completely avoided. We revisited the algorithm presented by Quaglia in this work and extended the probabilistic decision they made to create checkpoints. The main difference of our strategy is that they use a granularity of the intermediate events since the last saved checkpoint, while we utilize the granularity since the last rollback.

Kumar [Kumar et al. 2010] addressed the problem of minimizing the number of communication messages and checkpointing overhead in mobile distributed systems. They proposed a minimum-process coordinated checkpointing algorithm to avoid useless checkpoints and imposing low memory usages and computations overheads without blocking of processes. However, their strategy is application specific, and its deployment on a simulation framework is not straightforward and still could result in a high overhead of coordination messages. In this paper, we focus on checkpoint strategy for distributed and heterogeneous architectures of simulation.

In [Carvalho 2015] the authors presented an uncoordinated checkpointing algorithm to create checkpoints and its deployment on DCB. The algorithm allows each component of a simulation model to generate checkpoints independently according to a periodic time interval. The algorithm ensures a low management overhead, however, it produces a large number of useless checkpoints and consequently exhausts memory in large simulations. In this work, we integrated a probabilistic approach to scaling the execution of large simulations on DCB.

The contributions of Saker and Agbaria [Saker and Agbaria 2015] presented a checkpoint protocol which coordinates only with the processes that it has communicated with since the last checkpoint. The goal is to reduce the coordination effort as well as the checkpointing frequency. In this work, we also based the communication patterns to determine the checkpoint creation, but we do not use any coordination for the measurements.

In this work, we focus on a checkpoint approach for distributed and heterogeneous architectures of simulation. Our strategy does not require coordination to compute its metrics. Therefore, the strategy is suitable for coordinated and uncoordinated methods to minimize the number of inconsistent checkpoints.

6. Conclusions

Checkpoint strategies are essential to synchronize components of distributed simulations. However, checkpoint strategies turn to record useless checkpoints, which decrease the performance of the simulation. In this work, we presented metrics to measure the usability of checkpoints at runtime. Our approach analyzes the granularity of events since the last rollback to probe if the next checkpoint can be useful for the simulation.

We applied our metrics to extend a probabilistic decision, initially proposed by [Quaglia 1999], and deployed it on DCB. Experimental results presented an equilibrium between the number of checkpoints created and the time spent on rollbacks when compared with more traditional strategies. It means that our method established the more critical checkpoints for the rollbacks performed by the simulation. As a general result, our approach improves simulation performance by reducing the number of inconsistent checkpoints and consequently decreasing the time spent on rollbacks.

As future work,our method could be used to determine optimal intervals between checkpoints. In an architectural view, we also consider integrating coordination messages to improve the accuracy of our measurements and detect useless checkpoint efficiently. Moreover, we believe that our metrics can be utilized by garbage collectors to determine global states and remove inconsistent checkpoints.

Referências

Bouguerra, M.-S., Trystram, D., and Wagner, F. (2013). Complexity analysis of checkpoint scheduling with variable costs. *IEEE Transactions on Computers*, 62(6):1269–1275.

Carvalho, F. M. M., M. B. A. (2015). Hybrid synchronization in the dcb based on uncoordinated checkpoints. *Proceedings of ESM' 2015.*

Elnozahy, E. N. M., Alvisi, L., Wang, Y.-M., and Johnson, D. B. (2002). A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3):375–408.

Fagin, R., Fagin, R., Fagin, R., and Halpern, J. Y. (1994). Reasoning about knowledge and probability. *J. ACM*, 41(2):340–367.

Fu, D., Becker, M., and Szczerbicka, H. (2013). On the potential of semi-conservative look-ahead estimation in approximative distributed discrete event simulation. In *Proceedings of the 2013 Summer Computer Simulation Conference*, SCSC '13, pages 28:1–28:8, Vista, CA. Society for Modeling &#38; Simulation International.

Jefferson, D. R. (1985). Virtual time. *ACM Trans. Program. Lang. Syst.*, 7(3):404–425.

Johnson, D. B. (1990). *Distributed System Fault Tolerance Using Message Logging and Checkpointing*. PhD thesis, Houston, TX, USA. AAI9110983.

Kumar, S., Chauhan, R., and Kumar, P. (2010). A low overhead minimum process global snapshop collection algorithm for mobile distributed system. *arXiv preprint arXiv:1005.5440*.

Kunz, G., Stoffers, M., Gross, J., and Wehrle, K. (2012). Know thy simulation model: Analyzing event interactions for probabilistic synchronization in parallel simulations. In *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, SIMUTOOLS '12, pages 119–128, ICST, Brussels, Belgium, Belgium. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565.

Mattern, F. et al. (1989). Virtual time and global states of distributed systems. *Parallel and Distributed Algorithms*, 1(23):215–226.

Netzer, R. H. B. and Xu, J. (1995). Necessary and sufficient conditions for consistent global snapshots. *IEEE Trans. Parallel Distrib. Syst.*, 6(2):165–169.

Quaglia, F. (1999). Combining periodic and probabilistic checkpointing in optimistic simulation. In *Proceedings of the Thirteenth Workshop on Parallel and Distributed Simulation*, PADS '99, pages 109–116, Washington, DC, USA. IEEE Computer Society.

Reynolds, Jr., P. F. (1988). A spectrum of options for parallel simulation. In *Proceedings of the 20th Conference on Winter Simulation*, WSC '88, pages 325–332, New York, NY, USA. ACM.

Saker, S. and Agbaria, A. (2015). Communication pattern-based distributed snapshots in large-scale systems. In *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*, pages 1062–1071. IEEE.

Sato, K., Maruyama, N., Mohror, K., Moody, A., Gamblin, T., de Supinski, B. R., and Matsuoka, S. (2012). Design and modeling of a non-blocking checkpointing system. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 19:1–19:10, Los Alamitos, CA, USA. IEEE Computer Society Press.

Wang, Y., Gao, S., Jia, Z., and Li, X. (2009). Make a strategic decision using markov for dynamic checkpoint interval. In *Proceedings of the 2009 Ninth IEEE International Conference on Computer and Information Technology - Volume 02*, CIT '09, pages 197–202, Washington, DC, USA. IEEE Computer Society.